Clasificando texto con Python

Yuri Matheus
23/04/2021

Enseñaremos a nuestro programa qué comentarios están permitidos y cuáles no. Es decir, podemos darle muchos comentarios diciendo cuáles son ofensivos o no. Con esto, cuando aparezca un nuevo comentario, sabrá clasificarlo.

Enseñando a la máquina

Ya tengo un archivo aquí en mi computadora **CSV** con varios comentarios ya clasificados para enseñar nuestro algoritmo. Dado que vamos a utilizar este archivo, necesitamos abrirlo

.Una forma de abrir este archivo es usar la biblioteca **Pandas**. Esta es una biblioteca muy utilizada por personas que realizan análisis de datos o aprendizaje de máquinas, ya que proporciona una serie de funciones y clases que nos ayudan a manipular los datos.

Podemos instalar esta biblioteca con el comando pip:

```
\> pip install pandas
```

Ya tenemos Pandas instalados en nuestra computadora. Vamos a decirle a Python que comience a usarlo en nuestro código. Entonces le dijimos a Python que lo importara (`import`). Debido a que Pandas es una biblioteca ampliamente utilizada, es casi una convención importarla como (`as`) pd.

```
import pandas as pd
```

Digamos a Pandas que lea nuestro archivo de comentarios (`pd.read_csv()`) y asignarlos a una variable:

```
import pandas as pd
comentarios = pd.read_csv('comentarios.csv')
```

En este archivo CSV, tenemos los comentarios y sus clasificaciones. Para enseñar nuestro algoritmo, necesitamos pasar estos datos por separado, es decir, tenemos que decirle cuáles son los comentarios y cuáles son sus clasificaciones. Como necesitaremos estos valores, asignaremos a cada uno en una variable:

```
import pandas as pd
comentarios = pd.read_csv('comentarios.csv')

textos = comentarios['comentarios']

clasificacion = comentarios['clasificacion']
```

Cuando vayamos a enseñar nuestro algoritmo, tomará cada texto y clasificación y aprenderá si ese comentario es ofensivo o no... ¿Cada texto? Lo que hace que un comentario sea ofensivo es el comentario en sí, es decir, ¿el texto puro o las palabras que contiene?

Cada **palabra en un comentario tiene un peso**, es decir, no podemos ver un comentario como un todo, sino cada palabra que lo compone. Entonces, digamos a Python que tome esta nuestra string (`str`) de comentarios y la separe (`slipt()`):

```
import pandas as pd

comentarios = pd.read_csv('comentarios.csv')
textos = comentarios['comentarios']
clasificacion = comentarios['clasificacion']
textos_descompuestos = textos.str.split()
```

Para enseñar nuestro algoritmo, necesitamos contar cuántas veces ha aparecido cada palabra en un comentario. Es decir, de nuestro conjunto de palabras, tenemos que contar cuántas veces aparece cada palabra. Pero ¿ya tenemos ese conjunto de palabras?

Lo que tenemos hasta ahora es el texto de los comentarios descompuesto en palabras. Podemos decirle a Python que cree un conjunto (`set`) en `palabras` y por cada palabra en los `textos_descompuestos`, actualice este conjunto:

```
import pandas as pd

comentarios = pd.read_csv('comentarios.csv')
textos = comentarios['comentarios']
clasificacion = comentarios['clasificacion']
textos_descompuestos = textos.str.split()
palabras = set()
for palabra in textos_descompuestos:
    palabras.update(palabra)
```

Un **conjunto**, o un `set`, es una estructura de datos que contiene solo una ocurrencia de cada objeto, que en nuestro caso es una palabra. Es decir, si tenemos diez comentarios con la palabra `genial`, en nuestro conjunto, esa palabra aparecerá solo una vez.Pero si nuestra palabra solo aparece una vez en nuestro conjunto, ¿cómo sabemos cuántas veces ha aparecido?

Tenemos que contar cuántas veces ha aparecido cada palabra en el texto, podemos crear una función para eso. Nuestra función creará una lista con la cantidad de veces que apareció cada palabra en el comentario.

Para contar las palabras, necesitamos saber cuál es la palabra. Es decir, podemos usar la posición de nuestra palabra en el conjunto:

```
*# restante del código*

print(palabras[0])
```

Cuando ejecutamos este código, recibimos un error:

```
/home/yuri/Envs/ml/bin/python /home/yuri/Programas/Python/Classificação/classifica_comentarios.py
Traceback (most recent call last):
  File "/home/yuri/Programas/Python/Classificação/classifica_comentarios.py", line 13, in <module>
    print(palavras[0])
TypeError: 'set' object does not support indexing

Process finished with exit code 1
```

Nos dice que el objeto `set`, no admite la indexación. Es decir, no podemos acceder a un conjunto a través de un índice, como hacemos con las listas. Esto se debe a que los conjuntos tienen una forma ligeramente diferente de almacenar elementos.Los conjuntos utilizan una estructura llamada tabla de dispersión para almacenar sus datos.

Cuando agregamos un nuevo elemento, el conjunto usa este algoritmo para colocar nuestro dato en una posición. Por tanto, no podemos acceder a los datos de un conjunto por su índice, ya que no sabemos en qué posición se encuentra este elemento.

Entonces, ¿cómo contamos las palabras si no tenemos un índice?

Creando un traductor

Podemos crear un índice para nuestras palabras, es decir, para cada palabra de nuestro conjunto, podemos asignar un número.

Por ejemplo, podemos decir que la palabra `Gustó`tiene el índice `1`, permaneciendo: `('Gustó', 1)`. Pero, ¿tenemos que hacer esto para cada palabra?

Podemos crear una función que se encargue de esta tarea, sin embargo, Python nos proporciona muchas funciones, incluso una que hace exactamente lo que estamos buscando.

Queremos crear un índice para nuestras palabras, una forma de hacerlo es usando la función`range`. Así logramos crear una secuencia de números, por ejemplo, podemos decirle a `range` que cree una serie de números con base en el tamaño (`len`) de nuestro conjunto de palabras:

```
*# restante del código*

indices = range(len(palabras))
```

Si imprimimos esta variable, obtenemos el siguiente resultado:

```
/home/yuri/Envs/ml/bin/python /home/yuri/Programas/Python/Classificação/classifica_comentarios.py
range(0, 383)

Process finished with exit code 0
```

Nos devuelve una función `range` que va desde `0` hasta `383` exclusive. Podemos usar esto y asignar un número a cada palabra. Es decir, empaquetamos (`zip`) cada palabra con un número:

```
*# restante del código*

indices = range(len(palabras))zip(palabras, indices)
```

Con nuestras palabras agrupadas con un número, podemos crear un traductor, es decir, un diccionario para cada (`for`) palabra, indice en nuestra agrupación (`zip(palabras, índices)`):

```
*# restante del código*

indices = range(len(palabras))

traductor = {palabra: indice for palabra, indice in zip(palabras, indices)}
```

¡Genial! Ahora que cada palabra de nuestro traductor tiene un índice, finalmente podemos empezar a escribir nuestra función para contar las palabras.

Vectorizando un texto

Nuestra función tendrá que tomar todas las palabras de nuestro traductor y darnos una lista con el conteo de cada palabra en ese comentario. Para eso debe recibir un texto, es decir, el comentario, y nuestro traductor:

```
*# restante del código*

def vectorizar_textos(texto, traductor):
```

Nuestra función necesita contar cada palabra que aparece en el comentario, para eso podemos crear una lista del tamaño de nuestro traductor:

```
*# restante del código*

def vectorizar_textos(texto, traductor):  vector_de_palabras = [0] * len(tradu
```

Y, para cada palabra, comprobamos si está en nuestro traductor, si es así, tomamos la posición de esta palabra y aumentamos el valor en esa posición de nuestro vector:

```
*# restante del código*

def vectorizar_textos(texto, traductor):
    vector_de_palabras = [0] * len(traductor)
    for palabra in texto:
        if palabra in traductor:
        posicion = traductor[palabra]
        vector_de_palabras[posicion] += 1
    return vector_de_palabras
```

Ya tenemos la función de vectorizar nuestros textos. Ahora podemos hablar de nuestro script principal para vectorizar cada comentario, en nuestros `comentarios_descompuestos` usando nuestro traductor:

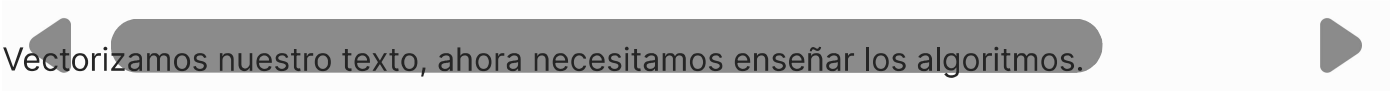
```
*# restante del codigo*

comentarios = pd.read_csv('comentarios.csv')
textos = comentarios['comentarios']
clasificacion = comentarios['clasificacion']
textos_descompuestos = textos.str.split()
palabras = set()

for palabra in textos_descompuestos:
    palabras.update(palabra)

indices = range(len(palabras))
print(indices)
```

```
traductor = {palabra: indice for palabra, indice in zip(palabras, indices)}
vectores_de_texto = [vectorizar_textos(texto, traductor) for texto in textos_d
```



Creando nuestros modelos

Ya tenemos nuestros textos vectorizados, ahora tenemos que entrenar nuestro algoritmo. Hay varios algoritmos que podemos utilizar. Un algoritmo muy utilizado cuando queremos clasificar algo es el **Teorema de Bayes**. Podemos usar el módulo **naive_bayes de la biblioteca scikit-learn**.

```
*# restante del codigo*

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
```

La clase **MultinomialNB** es una implementación del algoritmo de Bayes. Podemos usarlo para entrenar (fit) nuestro modelo:

```
*# restante del codigo*

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(vectores_de_texto, clasificacion)
```

Genial, nuestro modelo está entrenado, ahora podemos decirle que prediga (`predict`) un comentario, simplemente separe cada palabra del texto y vectorícelo.

```
*# restante del codigo*

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
```

```
modelo.fit(vectores_de_texto, clasificacion)
modelo.predict(comentario_vectorizado)
```

El método `**predect**` nos devuelve una array de Numpy con nuestra clasificación.

Para saber más

Clasificar comentarios es solo una tarea que podemos hacer con el aprendizaje de máquina. Con un código similar al que realizamos, podemos clasificar si un mensaje es o no spam. Podemos comprobar si el texto se trata de un tema financiero o no, entre otras varias posibilidades.

Este algoritmo nuestro ya funciona muy bien, pero podemos mejorarlo. Si echamos un vistazo a nuestro conjunto de `palabras` veremos algo como:

```
{'bueno', 'Bueno', 'automoviles', 'automovil', ...}
```

`Bueno` y `bueno` son las mismas palabras. La diferencia es que uno está en mayúsculas y el otro no. Porque Python es un lenguaje `case-sensitive`, distingue letras entre mayúsculas y minúsculas, y puede afectar la forma en que nuestro algoritmo clasifica los datos.

Para solucionar esto, podemos convertir todas las letras a minúsculas en el momento en que estemos descomponiendo los textos, por ejemplo:

```
import pandas as pd

comentarios = pd.read_csv('comentarios.csv')
textos = comentarios['comentarios']
clasificacion = comentarios['clasificacion']
textos_descompuestos = textos.str.lower().str.split()
palabras = set()

for palabra in textos_descompuestos:
    palabras.update(palabra)
```

```
*# restante del codigo*
```

Si miramos nuestro conjunto de palabras ahora, veremos que nuestro conjunto contiene solo una palabra bueno:

```
{'bueno', 'automoviles', 'automovil', ...}
```

Pero todavía tenemos algunas palabras en plural, ¿cómo podemos solucionar esto?

Podemos pasar de una palabra a otra para comprobar si es plural o no, pero eso es un poco laboroso. Otra forma es utilizar una biblioteca para transformar nuestras palabras en una, es decir, extraer la raíz de la palabra.

Una biblioteca ampliamente utilizada para esto es la **NLTK**. Esta biblioteca nos permite manipular textos de varias formas. Podemos extraer la raíz de las palabras, eliminar puntuaciones e incluso comprobar y eliminar términos comunes, como artículos y preposiciones.

Es decir, podemos mejorar mucho nuestro algoritmo eliminando "suciedades" que pueden entorpecer nuestro modelo en la parte de aprendizaje y predicción.

Aquí en **Alura**, tenemos una formación en Data Science. Allí aprenderás cómo funciona alguno de los algoritmos de clasificación, cómo usar la biblioteca NLTK y muchos más.

Puedes leer también:

- Buscando tweets con Python
- ¿Cómo comparar objetos en Python?
- Manipulando datos gigantes con Pandas

ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Lógica de Programación | Java

Cursos de Front End

HTML y CSS | JavaScript | React

Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps

Docker | Linux

Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento